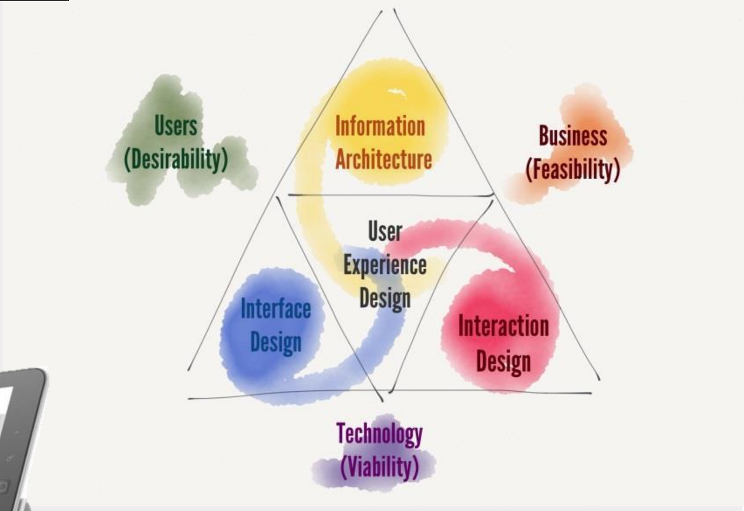
Friends of Oracle and Java
AMIS
Edisonbaan 15
Postbus 24
3430 AA Nieuwegein
T +31 30 601 60 00
E info@amis.nl
I   amis.nl

Whitepaper User Experience Frameworks

- Future of optimal UI development -



Authors
Robbrecht van Amerongen
Paco van der Linden

Date
July 2014

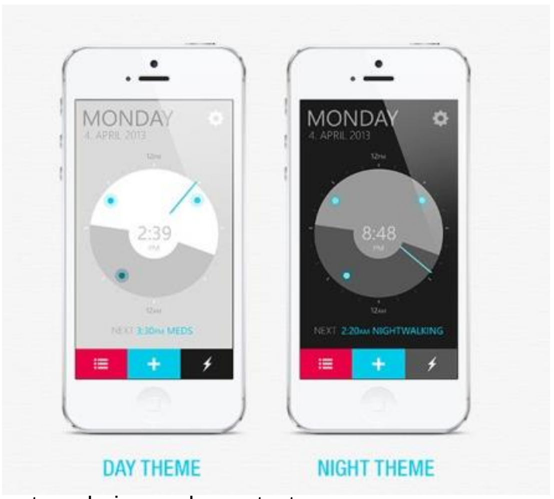
Table of Content

1 **What is user experience (UX)? 4**
1.1 Effective 4
1.2 Intuitive 4
1.3 Anticipating 4
1.4 Autonomous / personalization 5
1.5 Defaults 5
1.6 Grouping 5
1.7 Effective color usage 5
1.8 Reactive 5
1.9 Timely 6
1.10 Trustworthy 6
2 **Requirements for modern business UX 7**
2.1 User Centric 7
2.2 Device independent 7
2.3 Fast response 7
2.4 Scalable 7
2.5 Short development lifecycle 7
2.6 Decoupling of back end systems 7
2.7 Rich user experience 8
2.8 Ability to work offline 8
2.9 Mashability 8
3 **One size does not fit all 9**
3.1 Architectural models for modern user experience 9
3.2 Thin Client, Server side processing 9
3.3 Thin Server, Client side processing 9
3.4 Global difference in these models 9
3.5 Hybrid situation 12
3.6 Multiple small thin server apps combined in the client 12
3.7 Coupling and integrating thin server systems with backend systems 12
4 **Use case typologies 13**
4.1 Electronic banking application 13
4.2 News website 13
4.3 Customer Retail Shop 14
4.4 Logistics information provider 14
4.5 Extension on CRM system such as SAP or JDEdwards 15
4.6 Document management 15
4.7 Combining information from different sources 15
5 **Characteristics to consider for a modern UX framework 16**
5.1 Functionality 16
5.2 Technology 16
5.3 Development and deployment 16
5.4 Maturity and support 16
5.5 When to use which architecture 17
5.6 Several frameworks to consider in these architectural models 18

AMIS

Preface

" There's A Lot More Behind This Pretty Face "



Modern business web applications are faced with rapidly changing requirements. Users can choose from a wide variety of systems and have a distinct preference when it comes to usability. The forced or required use of one single system is becoming unacceptable. So are systems with poor user experience, even if the business logic behind it is implemented well. Business users demand apps that are effective, intuitive and efficient. They must have fast performance and 24/7 availability. And they have to look sexy…..

User Experience (UX) has become the major reason for rejecting a system during end user tests or even worse: after go-live. Users have high expectations, based on the frequent use of social media applications, and expect the same standard for their own business systems. Users expect an easy to use interface, fast interface response time, usage on a variety of different devices, easy login and offline availability.

To be able to meet these expectations, software developers require short development cycles and full test coverage to support agile development cycles, seamless support for multiple platforms and devices, secure transactions and easy decoupling from backend systems. And during operations, systems managers, need to be prepared for the unpredictable timing and growth of the visitors of business applications. In some cases the system and hosting platforms need to be able to support a burst in demand or the exponential growth of the user community without drastic changes to the application architecture.

This also requires a productive development environment with massive scalability for both the number of developers and eventually the number of concurrent end users. Frameworks with an intrinsic agile capability to modify and expand the functionality with a very short time to market.

We feel there is no one-size-fits-all solution for UX requirements. We see a shift from technology derived designs towards user centric designs facilitating every end user with a personalized, timely, effective interface. This kind of approach will lead to more effective, easy to use and enjoyable applications.

This whitepaper gives an overview of user experience guidelines. These guidelines translate to additional UX requirements when designing and building a new user interface on modern systems. We will also discuss the two major architectural paradigms for user interface development, followed by an overview of the major frameworks and technologies used for implementing this architecture.

Finally we will give a number of business examples and the preferred technology for implementing the requirements.

1 What is user experience (UX)?

User experience is not only about graphic design. User Experience enables end users to effectively do their job. A good user interface design is a small part of a successful user experience. The interface defines the 'face' of the application and the user experience defines the impression the application makes on the end user.

Successful user experience leaves a pleasant impression with the user when using the application. A successful UX is often taken for granted, while a bad UX is noticed immediately. Therefore it is advisable to include UX verification and UX testing as a formal activity in your application development process.

Guidelines for successful UX

A successful user experience has to be aimed at the user of the application and not the product itself, the developer or the organization offering the application. The major guideline is to have a simplified user experience for the end user. Keep the following guidelines in mind to achieve a successful UX:

1.1 Effective

A successful UX makes the life of the user easier. It presents the right content prepares steps. It reduces unnecessary searching information in a vast amount of data. It enables the user do more work in less time.
An effective user interface saves time and reduces user errors resulting from incorrect usage of the application. The accompanying picture gives an example of two different interfaces of a blender.

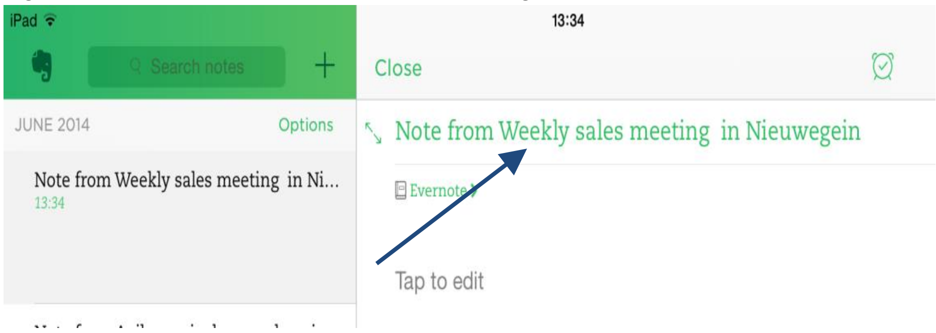
1.2 Intuitive

Intuitive means that a user interface shows what the user expects. So that there is no need to read the manual. Users are able to work with the application after only a short instruction that is focused on the business side of the solution. It also means that the software has to be able to adapt to the experience level of the user. Where new users need to be guided through the interface, experienced users focus on effectiveness usage, so tips and hints can disappear after a while. The NEST interface (accompanying picture) is very intuitive. The color shows if it is cooling or heating and the dial gives you only two options: increase or decrease the temperature.

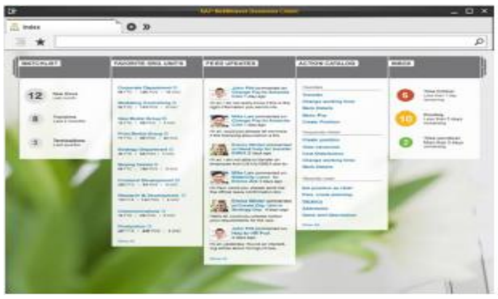
1.3 Anticipating

Make full use of the information that is available of a user. Provide enough context, so that the application can anticipate the user's needs. This means using flexible presets and pre-selections. Use context information (like location, agenda, goals, activities of peer users and organizations) and hide items that are less frequently used to show only the information that adds maximum value for the user. Examples: when a user creates a meeting minute, propose the title from a recent meeting from the calendar of the user; when searching for customers show the top 10 customers based on turnover and number of contacts.

The Evernote interface anticipates when creating a new note: the default title is the name of a recent calendar meeting concatenated with the city name derived from the GPS coordinates from the place where the note is taken.

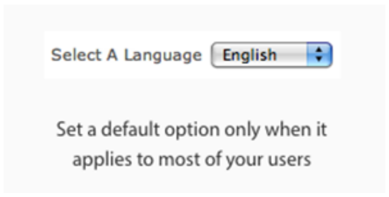

iPad
13:34
Search notes
Close
JUNE 2014
Options
Note from Weekly sales meeting in Ni...
13:34
Note from Weekly sales meeting in Nieuwegein
Evernote
Tap to edit

1.4 Autonomous / personalization



Users must be empowered to customize the user interface experience to their ‘own’ preferences. It is important the users consider the UX as something they own and not as something that has been forced on them. This applies to color, logo and text size and font. Also support customizing the ordering of menu items, grouping of icons and items displayed on screens.. See example of a user dashboard on the right.

1.5 Defaults

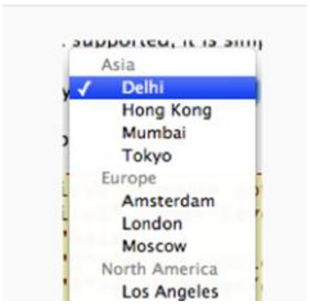


Select A Language English

Set a default option only when it applies to most of your users

The use of defaults will maximize the effectiveness of the user. By anticipating on the users behavior and making the interface intuitive, the defaults can be set to support the desired application behavior for the user. This can be done by assessing some easy defaults. Examples: an event registration will anticipate that the default number of participants is 1 and not 0. When requesting a report of revenue information, the default timeframe is last month and not the current month. In the example to the right, the default language is derived from the language preference in the users’ browser.

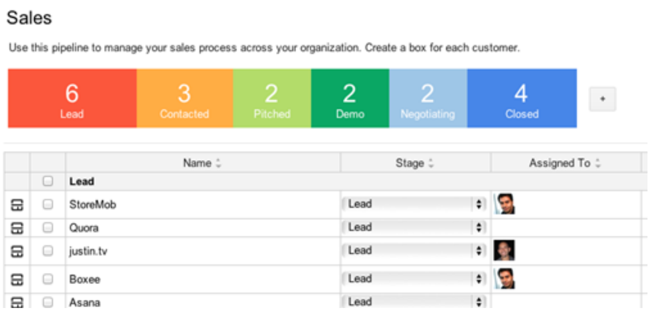
1.6 Grouping



Asia
✓ Delhi
Hong Kong
Mumbai
Tokyo
Europe
Amsterdam
London
Moscow
North America
Los Angeles

Related information needs to be presented side by side. So the users are not distracted or spend too much time searching for the right information. When you group similar information, users do not have to switch back and forward to all pieces of the content. Another example of groupings can be found in large drop down lists (example on the right)

1.7 Effective color usage



Sales

Use this pipeline to manage your sales process across your organization. Create a box for each customer.

6 Lead	3 Contacted	2 Pitched	2 Demo	2 Negotiating	4 Closed

Name	Stage	Assigned To
Lead		
StoreMob	Lead	
Quora	Lead	
justin.tv	Lead	
Boxee	Lead	
Asana	Lead	

Effective use of colors increases the involvement of a user It gives an indication of the purpose of the application. Example: Use orange for a sales and action driven items and green for a registration service. Google “Color Theory” for more information.

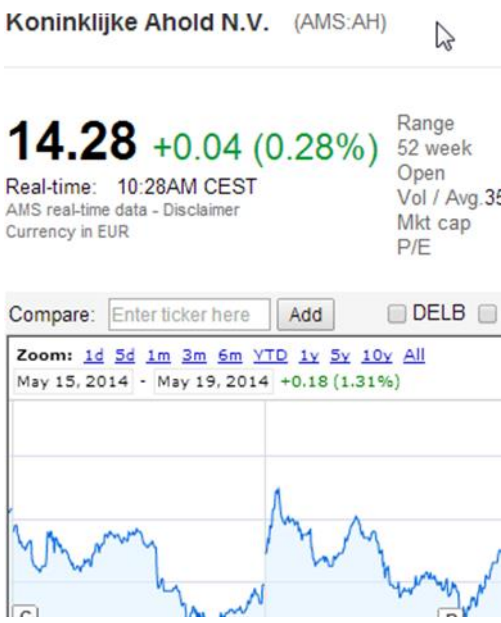
1.8 Reactive

An interface needs to adapt to the device you are using. If you are using the application on a big display, all information can be displayed, including images and menu. When you are using the system on a device, the design adjusts to the smaller screen and shows the information differently. Adaptive design is not only focused on rearranging the layout. This form of design demands prioritization in the information that is presented. Less important information is being pushed to a lower part or is hidden on small screens. Your development environment has to support this kind of adaptive designs, so that you do not have to build the same interface several times.

1.9 Timely

Timely can be explained in two ways. Firstly, by performance of the system, the sheer reaction time. Users do not want to wait more than 2 seconds for their request to be processed. Secondly, the timeliness of the content. Present content on the moment the user needs it. Make use of the context like geo location, other actions and the actual date and time. Examples: show past contact notes when you are at the geo location of your customer. Show only the notes of this customer. Show orders needing action when their delivery time is about to expire. Like the oil indicator in your car that only shows a warning when you need to take action. You do not see a green oil indicator when the oil level is ok.

1.10 Trustworthy


Koninklijke Ahold N.V. (AMS:AH)

14.28 +0.04 (0.28%)
Real-time: 10:28AM CEST
AMS real-time data - Disclaimer
Currency in EUR

Range
52 week
Open
Vol / Avg.3
Mkt cap
P/E

May 15, 2014 - May 19, 2014 +0.18 (1.31%)

The end user needs to experience the application as trustworthy. Especially when working with confidential information like financial or medical data. A professional look and feel helps. Also pay attention to the origin of the information and the value other users give to this information. By displaying the source of the information and a timestamp you can validate its value (for example for currency rates: "Source : Yahoo Finance: 2014-04-12 09:22PM"). Also display the comments and opinion of other users. You can do this via a rating system or by showing comments on specific information items.

2 Requirements for modern business UX

The difference between a modern user interface for business environments and for the consumer market is starting to fade. People expect business apps to work in the same way as the systems they use for their personal activities. Thus we can merge the requirements of these two kinds of interfaces.

2.1 User Centric

Most user interfaces are designed around the product or service itself, the supplying organization or the developers of the applications. However, new user interfaces require a user centric approach. The system must be aware of the context, interests, goals and responsibilities of the user and implements this in the user interface. Users demand a my.organization.com environment. An interface aimed at the user and that displays the tasks, products, messages, customers and orders of that particular user. And not just all of the available content, tasks, products, messages and orders. This is vital to new user interfaces: Only show to the information that is important in the context of current specific user.

2.2 Device independent

Users demand applications that operate on all the devices they use and they also want consistency; personalization should apply to each device they use. It should not be limited to the office infrastructure. Working from home and Bring Your Own Device are trends that require systems to operate on a wide variety of devices, browsers and operating systems.

2.3 Fast response

Users expect a timely response of the application. For both the interaction with the user interface and the time it takes to present information from the systems resources (database). Make sure the user gets timely feedback from the application by showing the effect of their actions. By executing the action immediately or by showing a timer or progress bar.. If the application is lacking immediate action feedback, the user will get irritated or confused and submit the same action again and again.

2.4 Scalable

The times when there was a steady and predictable flow of users of your application have gone. Almost all systems must be prepared for peak loads. Due to seasonal influences, additional popularity of certain items or just rising reporting demands. Offering applications via a cloud based infrastructure might resolve some of these peak loads, but the solution is using the appropriate architecture for scalable solutions

2.5 Short development lifecycle

Short time to market and fast response on changing business requirements is essential for a modern user interface. This requires a toolkit with a short development lifecycle, supporting agile software development. The development toolkit needs to support quick development, prototyping, be equipped with a fast test framework and it must be manageable for a large group of developers. The time between development and deployment should be as short as possible.

2.6 Decoupling of back end systems

A user interface has a shorter life cycle than back end systems. The interface has to respond and adapt to fast changing demands of stakeholders and users. The expertise and technology for designing and creating a user interface is also different to that required for developing back end systems. Effective development and fast response to changes is possible when there is a clear decoupling of the user interface from the back end system. This way your web designers can create a user experience without having to bother with the possibilities or limitations of the interface services connecting the back end system.

2.7 Rich user experience

Users expect a rich user experience from (web) applications. This means instant feedback on data manipulation, drag and drop functionality, visualization with extensive charts and (info)graphics and animations to grasp attention and give feedback to the user. The user interface offers information in a more condensed manner and responds to the user by expanding or reducing information based on the users interaction with the system.

2.8 Ability to work offline

Working offline is an often requested feature of user applications. Even with good mobile coverage and availability of wireless connection points there is still a need for working offline. Often to bridge temporal loss of network connection when switching networks or when the user is temporarily in a networking blackspot. Users expect the system to keep working during these moments and catch up automatically when the network connection restores. This means a local copy of the edited data needs to be stored in the client that synchronizes the moment the connection restores. Data security is also important, since users expect their sensitive data to be secure on the local software. The ability to work offline is an architectural requirement. Since user sessions with the system can stop and start at any time, the server needs to be able to handle this.

2.9 Mashability

Usually applications offer the combination, enrichment and filtering of data in the backend system. A Service Oriented Architecture will enable the application to combine several sources to one functional data stream on the server side. In some architectures there is less need for combining this information since the data source is publicly available. You can then use a mashup to enrich data from your private company system with public data. The data is combined in the client, resulting in faster response and reduced load on the server. Examples: combining the visitors address of your customer with public map data (including traffic information); the invoice amount in a default currency combined with public currency rates to display the amount in different currencies or combining private research information with public news sources on the same subjects. A mashup offers the possibility to include real time public data sources in your application and offer extra information to your user. Mashup will not require additional server resources since the mashup is done in the client interface (web browser). An user experience can be composed of several client side mashup applications. These applications can share resources, making it possible to define a uniform skin, language or other preferences.

3 One size does not fit all

When you starting to create an attractive, appealing user interface, you need to consider the best architecture and the best frameworks for your interface. There is no one size fits all solution, no silver bullet framework to support all needs and requirements of a modern demanding user for every type of business, market or application.
Use a customized approach and select the best architecture and frameworks for each situation. However, this takes time and experience.

3.1 Architectural models for modern user experience

There are two architectural models for modern user interface in enterprise applications; thin server and thin client. “Thin” refers to the size of the application and/or the hardware on either the server or the client. Thin server means the user interface is completely created in the client and thin client means most of the user interface is generated on the server. We decided not to discuss a third group of models comprised of scripted or database generated interfaces in this whitepaper. They are not very suitable for an enterprise architecture and they offer less usability features. These interfaces are build in languages like Oracle APEX and PHP. They are perfect for delivering user interfaces in a somewhat straightforward landscape but they do not offer the flexibility, scalability and user experience required for enterprise applications. Therefore, we do not consider them a valid alternative for application development for large scaled enterprise systems..

3.2 Thin Client, Server side processing

Thin client architecture is designed for systems requiring limited intelligence and processing power on the client side. The server generates the complete interface and sends it to the client. The client only has to render the interface. The majority of the processing is done on the server, so that there only limited requirement to the client hardware. This means you can use simple terminals as clients. The server is always aware of the actual state of the user sessions on the client and they can easily be transferred to other terminals. The server has total control over the actions of the user and these can be completely traced since all actions result in a callback to the server. This makes control over complex multi-step transactions very easy.

Thin clients work really well when little or no data can be stored on the client and the session of the user needs to be completely controlled by the server. Examples of these frameworks are Spring, JSF PrimeFaces / RichFaces and Oracle ADF Faces.

3.3 Thin Server, Client side processing

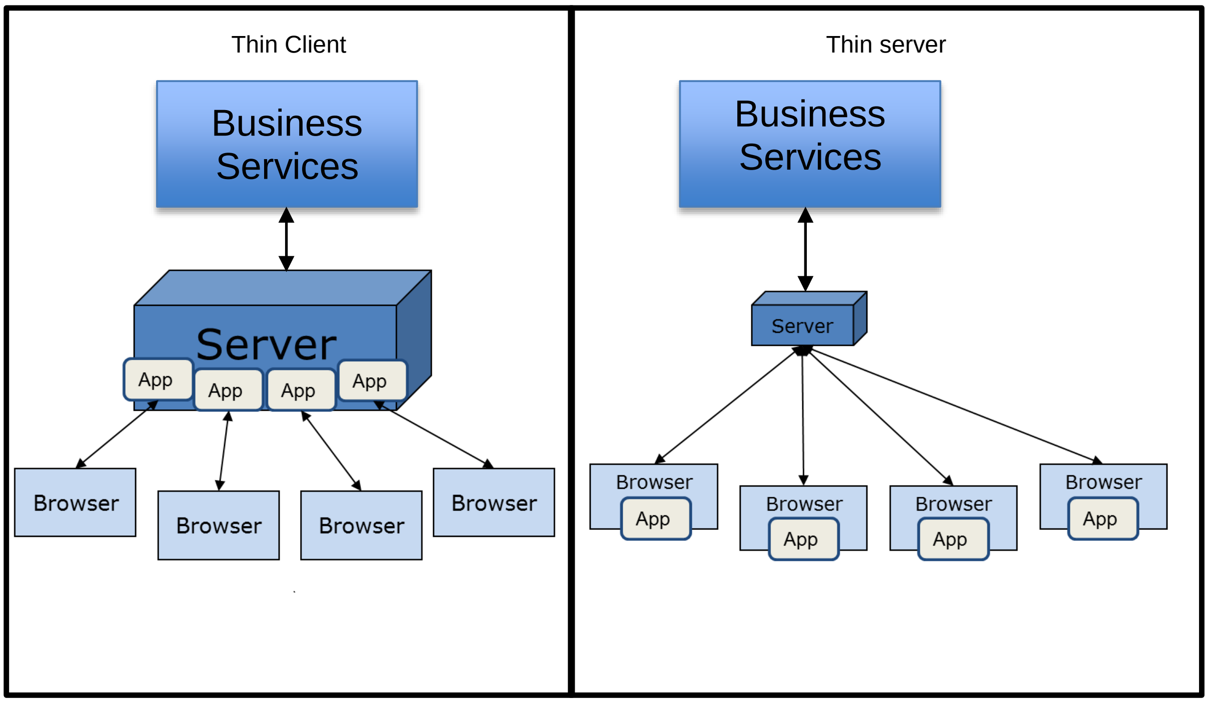
Thin server architecture is designed to bring the application as close to the user as possible. Most of the processing is done on the client side (e.g. the web browser of the user or the mobile app). The interface is rendered in the client by combining data with the user interface design and logic. Business logic is brought to the server side and interface logic to the client side. This is less complex since client and server development is done separately; Communication between client and server goes via an independent protocol and does not depend on the architecture of the server (eg SOAP or JSON). The thin server architecture moves session management and page rendering to the client side. This results in a radical reduction of server load and the possibility of massive scalability. In fact, you are utilizing the processing resources of the client and every new user brings hisown processing power (own client machine/browser). Examples of frameworks used to implement this architecture are Bootstrap, AngularJS, Polymer, GWT and Ember.

3.4 Global difference in these models

The difference between these frameworks is the location of the session state and the interface logic. Thin Client architecture keeps the session and logic on the server; Thin Server architecture keeps the logic and session on the client side (the “machine of the user”).

The advantages of Thin Server are fast response to user actions, ability to work offline, fine grained control over UI rendering, native CSS control and scalability. Thin server works well for systems with a lot of data read actions, systems with a requirement for fast response time and a fairly unpredictable number of users. The disadvantage of thin server is the abstraction of the server side code towards the frontend developer. The developer is not aware of the structure and capabilities of the backend system. This can also be explained as an advantage since the complexity is shielded from the developer.

The advantages of Thin Client are traceability of the users actions, suitability for complex data manipulations and complete server side control over the users session.
The disadvantage of thin client is server side session management. With a large number of concurrent users all sessions need to be stored on the server. In the case of failure, all session information needs to be transferred to another server within the same cluster. It takes a lot of memory and processing power on the server side.



An overview of the major differences between these two models:

Differences between Thin Client and Thin Server approach		
	Thin client	**Thin server**
Location of data and logic	All session data is kept at the server. All application logic is executed at the server. For a medium to large user base the handling of sessions often requires a significant investment in server hardware. A sudden increase of the number of users can be a challenge. Either a single server has to scale up the number of clients, or fairly complicated mechanisms need to be used to allow multiple servers to share the session data of all clients (scale out). These mechanisms include fast distributed session persistence and sticky load-balancing.	All session data is kept at the client side. The rendering of the user interface is executed in the client. Limited server side processing power is required for the interface, just for gathering the data. Rather simple server side architecture. Scaling is fairly straightforward by adding more servers with the same configuration. No session persistence needed on the server.
Session failover	Complicated mechanisms are needed to provide failover of servers for reliability. When a server goes down all the sessions that were handled by that server are lost unless all sessions are persisted after every request (mouse click)	No additional requirements for session failover. Just handling this via a simple load balancing configuration is sufficient. Sessions are not persisted on the server.
Performance and scalability	Performance optimization is rather challenging because (almost) every click results in a round-trip to the server where the corresponding session needs to be looked up and a new (partial) page needs to be rendered and sent to the client. This is even the case for clicks without side effect (no change in session state) or clicks that result in a page that has been rendered before.	Optimized for scalability since every new user brings its own part of the processing power (their own client). Ability to execute a big part of the navigation and logic in the client without any interaction with the server. Only the start and the end of the process will have impact on the server.
Handling of browser events	Browser events like back-button and refresh are complex to handle and often result in inconsistent session state or invalid pages.	Ability to easily respond to browser events like refresh and pressing the back-button without effect on the session state.
Session management	Session is always active. Session timeouts needed to prevent unlimited memory usage on the server.	No sessions on server. No need for allocating additional resources to store sessions on the server.
Upgrades and development cycle	Live upgrading of an application (i.e. while it stays online) is possible but the webserver needs to be able to manage the session state while performing the upgrade.	Live upgrade is easy. Especially in a multiple server solution with a load balancer. The next request is handled through the next updated server instance.
Offline usage	Because the session is kept at the server, it is not possible to use the application when the network connection is (temporarily) interrupted. Unless specific operations are built in to handle working offline.	Working offline is supported by the client side data store and client side session management.
Support for mobile	Because of the nature of mobile devices, this architecture is less suitable for mobile apps.	Framework is better suited for mobile app development since device specific rendering and working offline are easier implemented.

3.5 Hybrid situation

The two different models both have their advantages and disadvantages. A hybrid solution is an option when you want to profit from all advantages. Using a thin server architecture for fast reading and to prepare for unpredictable variations in the user load. Using a thin client when the representation of the data varies per device and the pages are primarily read only. Then these pages can be rendered on the server and cached for future usage. An example of this hybrid solution is an online job-search service with registration module. The search module can be developed in thin server architecture and the registration module in thin client architecture. From the point of registration the user gets more context sensitive information depending on his personal profile and this is rendered and generated on the server side.

3.6 Multiple small thin server apps combined in the client

Another option is to develop several small thin server applications (‘rich client’ applications). This makes the development process easier as the development of these applications can be assigned to separate teams. The applications are combined as a mashup in the client. Thin server architecture supports session management and storage on the client. Within the mashed up applications the session information can be shared between several thin server applications from the same supplier. So the user preferences, data and status can be shared between these applications. Avery flexible and scalable solution.

3.7 Coupling and integrating thin server systems with backend systems

How are thin server systems coupled to the backend systems? Most of the rendering of the screen logic and navigational logic is being processed by the client system. The server part is in most cases only the proxy with the services of the backend system. These services can be based on multiple combined sources too. In this architecture you need to choose where to locate the development and processing effort.

Most thin server / rich client systems are interacting via an open standard protocol. This results in a clear boundaries between the frontend development and the backend development. The interaction protocol is most often REST/XML or ar REST/JSON configuration. All systems providing these kind of standard interfaces can be used as a backend provider. Because of the clear boundaries the backend system is not limited to one specific technology, as long as the backend is capable of providing the data in fast and scalable. This enables the frontend to integrate with a service oriented architecture such as an enterprise service bus (ESB). Technologies like Oracle Service Bus / SOA Suite, ADF Business Components, Microsoft Biztalk, JAX-RS or Mule are able to support the specified formats.

4 Use case typologies

Below we describe several use case typologies. For each typology we describe the characteristics of the applications and the preferred general architecture for these kind of applications. Remember these are not actual customer cases, they are virtual, 'ideal' situations.

4.1 Electronic banking application

Banking: The ABC Bank is a large European bank. 80% of its consumer- and 90% business transactions are handled online. Clients look for ease of payment, integration and a multi-channel experience (pc, tablet, mobile). Delivering a secure environment is obligatory. That means that external users cannot modify messages during the transaction process. Users want an interface that is trustworthy and works seamlessly in different browsers, including the less obvious ones. Besides fast and secure transactions the scalability of the services is very important since the application usage van fluctuate drastically during the day. ABC Bank needs to offer a stable and reliable service.

Keywords: **secure, fast, reliable, user friendly, large developer group, regression test**

General architecture: the actual transactions are not very complex. The number of different types is limited. Reliability, volume and security are the most important aspects of this system. Most of the business logic resides at the server side or even in the backend system. The logic on the device is fairly simple. To support fast response, a reliable enterprise service bus is used for stable communication between frontend and backend. Due to the vast number of concurrent users, it is advisable to have the session reside on the client. The backend services are hidden via this mechanism. To support a clean frontend structure with fast response, a thin server architectures is most advisable. All security, access and tracing rules are implemented in the service bus and backend.

4.2 News website

Media: Medi@ is a news website where national and international news is combined. The news site has a large amount of daily visitors using PC, tablet and mobile. All news items include photo and video material, and users can contribute to the items by adding comments and media. Content syndication towards other news sites is also offered

Keywords: **fast, multi-channel, multimedia,**

General architecture: After being published, the news articles hardly change. Users read most articles only once. To ensure fast performance, caching of data and styling is important. In this scenario a thin client architecture is the best choice. Content can be cached with styling for different devices or even be perfected for the most frequently used devices. Almost all recent content can be placed as complete rendered pages in the servers memory. Due to SEO considerations it is advisable to keep the pages and structure of the content as stable as possible. Server side control will provide this stability.

4.3 Customer Retail Shop

Consumer Retail: Sphere is an online retail shop that focusses on a complete customer experience. They are not only selling products, but they also focus on aftersales and they give their consumers the opportunity to review products. Social sign is provided and it gives Sphere the opportunity to provide the user with product suggestions. The shop also has social media sharing options. Users use the application at home on their laptop for product browsing and also on their mobile phone to check prices while shopping in a physical shop.

Keywords: **webshop, secure, user centered, multiple devices, scalable**

General architecture: Customers of Sphere expect personalized content with push functionality when a newly desired product becomes available. Part of the content is general for all users and part of the content is personalized . The users will use multiple devices to access their accounts. Product prices will also vary between customers depending on their purchase history and loyalty programs. The best architecture Sphere is a combined or hybrid model. The generic information is stored as cached information on the server. The personal information including the shopping cart and wish list will be handled as a thin server application, so that the list is always saved on the server side to provide the same cart when the user logs in with another device. .

4.4 Logistics information provider

ShipIT is a large logistics company located in a seaport. Their monitoring application must be able to show a user friendly overview of all the goods, with different kinds of markings for monitoring purposes. Different departments consult the application for specific data and customers with exclusive access to their own data.It therefore has to be easy to filter data. The number of data entries is huge; ShipIT handles 50-100.000 shipments a day. The information about these packages needs to be accurate and timely. The size of the individual information packages is relatively small and most information requests come from small mobile devices.

Keywords: **large data, secure, multi user groups, lot of data, almost no updates from client, real time, fast response**

General Architecture: The use of the application is diverse; mobile users combined with desktop users. The information requests are also very diverse. The same information is accessed no more than two times a day. The number of requests is very high and the information size is relatively small. There are different groups of users who use the information differently (trend reporting or individual shipment). A thin server architecture with a high available service layer to comply with the different information needs is the most likely architecture. The service is connected with a rich client interface suitable to operate on different mobile devices. There is hardly a need for caching of data, since the data needs to be real time and is not shared among other users. The use of a data grid could be considered (for more information see HazelCast or Oracle Coherence). This grid can provide real time specific information on shipments without a full callback to the actual database.

4.5 Extension on CRM system such as SAP or JDEdwards

The Holding Inc. is a large conglomerate of construction stores selling to both consumers and building companies. They are based in 8 countries and have a total of 240 stores. The Holding has 8 distinct brands that are independent. All stores are managed by one central JDEdwards implementation that supplies them with logistical information and that reports monthly to the holding. The high-end stores offer additional value to their key customers by offering online services to integrate with their planning system. They also offer specific software to detect planning faults and to prevent ordering of incompatible material within the same project. These services are offered as a web-service and as web page interface. They are custom extension to the existing software.

Keywords: **integration, extendibility, scalability, customization**

The most important characteristic of this solution are extendibility, customization and resilience. Software updates of the CRM system should have no effect on the custom service. The architecture of these extensions consists of a service layer responsible for the decoupling of the CRM system and the custom software. The custom software consists of several logical modules providing the additional services. These modules will most likely reside on the server. The provided service layer for valuable customers is created with a webservice implementation such as Oracle SOA Suite or JaxWS. The web pages interface is a thin server implementation based on the same web services. This will facilitate a lean architecture on the server side and fast response for the specific customers.

4.6 Document management

DocIT is a project management organization involved in large projects. For every project they work with a fixed document structure where several employees, with different user rights, have access to during the project. PM is looking for a solution that allows different users to work simultaneously on the documents online, and also offline on their laptop or tablet. They also need to be able to contribute to the documents and make annotations. Some of the projects are under non-disclosure and data cannot leave the (virtual) office unsecured.

Keywords: **On- and offline, document management, secure, multi user rights,**

The most important characteristics are security and data storage. The number of users is known, so they are able to size the server for this requirement. Need for collaboration and secure storage of data is implemented by server side session management. A thin client architecture is the best option for this requirement .

4.7 Combining information from different sources

ZAPP energy is an energy and gas company providing end user services and supporting transportation. Their website is a mashup application consisting of Twitter and Google Maps. Based on geo tagged twitter massages like #gasleak they show the information on the map. Clients and employees can see this information on their internet device. This information is combined with information on actual planned work on the gas network so the operators of ZAPP are able to analyze if the public notifications correspond with actual planned maintenance work on the gas network.

Keywords: **Mashup, combination of private and public data, geo based data, real time**

General Architecture: This solution needs a mashup frontend architecture. Private data about planned maintenance data is combined with public data. The combination of these two sources is designed as a mashup application in the client application. This architecture is thin server.

5 Characteristics to consider for a modern UX framework

We made a short study and assessment of several frameworks to be considered when implementing a thin server or thin client architecture. This study can be a starting point for a new application since these technologies are optimized to work together. The frameworks are selected based upon four characteristics.

5.1 Functionality

Considering modern functionality, the framework has to support development for mobile platforms and have a considerable rich user interface. It has to support adaptive design and HTML5 / CSS3. Frameworks with pre-defined building blocks or components that can be extended and customized are preferable. The framework has to support fast performing applications and being able to cope with a lot of data.

5.2 Technology

The technology has to support modern interaction with backend systems using JSON, XML. Ideally, it should have default support for RESTful interfaces. The frameworks have to support test automation and powerful debug capabilities matching the possibilities of contemporary IDEs. Since we operate in the Java domain, our frameworks have to appeal to developers with a Java background. For this group of developers the learning curve has to be acceptable. The framework has to provide a clear pattern for dealing with enterprise application security.

5.3 Development and deployment

We require our frameworks to provide a short development cycle and short deployment cycle (edit-compile-deploy-run-debug cycle). It has to be able to deal with a collaborative development environment where a team of multiple developers modifies the complete codebase. It has to offer possibilities to prototype parts of the system and offer great debug support.

5.4 Maturity and support

The framework has to reach some point of maturity to ensure the continuity. This means a considerable number of implementations, involved developers and a fast response on questions or defects found in the framework. This is reflected in a considerable and mature user community possibly supported and facilitated by a commercial vendor. The framework has to offer a sensible licence model to enable us to develop and sell software with it without complicated legal consequences.

5.5 When to use which architecture

Below we describe a number of aspects to take into account when making a choice for a Thin Server/Rich Client or a Thin Client/Rich Server architecture. These are general considerations .The actual choice between these frameworks also depends on other factors like available licences, knowledge and experience of the development team and general company architectural guidelines. These other factors are not included in this list. It is certainly reasonable to have different architecture implementations within the same company. This depends on the competences and maturity of the development team and the desired solution for user experience.

Aspect	Thin server/Rich Client	Thin client/Rich Server
Number of users	Unknown number of users. Unexpected growth in public domain users.	Known number of concurrent users identified by their own login. Predictable number of anonymous users.
Usage	Public usage / public domain applications	Private usage / enterprise applications
Interaction	For both process oriented interaction and straightforward create, read, update, delete actions (CRUD). Saving the changes only at the last action.	For both process oriented interaction and CRUD. Intermediate update.
Scalability	Highly scalable. In general less complex.,	Scalability limited to server resources and clustering options. In general more complex.
Design	Design can be decoupled completely from the server side development.	More challenges to decouple the interaction design with the server side development. .
Development team	Clearer distinction between frontend and backend development team.	Frontend and server side development team overlap.
Development cycle	Very fast for frontend	Not so fast for frontend technology.
Grip on data and session	Session resides on client. Server is stateless and has no information about the session.	Session is on server and in complete control of the server.
Outage and Failover	Server outage does not reflect in client response until server data is needed. Failover on another server is just a matter of diverting the request to another server via the load balancer	Server outage is directly reflected in client outage. Failover is more complex since the complete session needs to be stored at the failover server. Cluster technology is suited to resolve this problem. This architecture can make the system complex and expensive.
Frontend	More complicated client development since data storage and session management has to be implemented in the client	Client development is less complicated.
Multi device support	Liquid and responsive design is generated client side. Support for multiple devices depends on the development team or the framework used.	Depends on the supported devices by the framework (or the development team).
Smoothness of the UX	The user interface rendering is done in the client. Hence the interface will be perceived as more smooth. All actions are executed in the same page.	Page refresh after most of the actions. This will be perceived as a less smooth user interface.
Learning curve	Steep learning curve	Average learning curve. Depends on the framework used
Available resources	Depends on the framework.	Depends on the framework.

5.6 Several frameworks to consider in these architectural models

There is a vast list of available frameworks and development languages. We limited ourselves to those frameworks used for professional enterprise applications. We excluded scripted frameworks, frameworks that are infrequently usage and frameworks that are still in a very immature development stage and. Our list presentsframeworks that are suited for our area of expertise: Oracle and Java Technology.

CSS Frameworks

CSS is just styling, no logic or server communication. These CSS frameworks combine HTML5 and CSS technologies in some very useful components. We prefer Bootstrap. Other choices are Topcoat and YUI.

- Bootstrap (OpenSource)
    - Combines HTML + CSS (+ some JavaScript)
    - Supports responsive design for multiple device formats.
    - Easy integration with other frameworks for frontend development.
    - Tailoring with JavaScript for integration with other frameworks.

CSS support: editing and modifying css structures

- LESS or SASS
    - Editors for modifying CSS structures. Configuration of Bootstrap is possible via parameters..

JavaScript libraries

These libraries are especially valuable for filling in the missing links between the JavaScript and other frameworks. They assist developers in fast development and hide the browser specific anomalies found in the use of JavaScript.

- jQuery
    - This is a fairly simple layer on the low-level browser DOM API. JQuery is commonly used as a layer on top of the low level browser DOM-API. The concepts of jQuery are also included in the DART language.
    - jQuery is often used to extend functionality that is not by default included in another framework.
- Alternatives: YUI (JavaScript) Bootstrap (JavaScript)

Client-side development languages

- Dart
    - Also known as the new JavaScript. Is actively developed and promoted by Google as the replacement for JavaScript. Dart needs it's own VM to run the code. A growing number of browsers are supporting native Dart execution. This number is probably going to grow for the upcoming months. Dart has a faster execution (about 5 times faster compared to JavaScript) and is therefore more energy efficient. This will result in a better battery performance for mobile devices. For browsers not able to support native Dart the fallback scenario is to use the JavaScript compiled output of Dart.
      Advantages compared to JavaScript:
        - Browser independent;
        - Clear and readable syntax (contrary to JavaScript);
        - Contains the best practice we learned from the using of jQuery;
        - Is, just like Java, very suitable to develop large scaled applications (This is very difficult in JavaScript).

      You can recognize most frameworks using DART since they carry the word in the framework name (AgularDart).
- Java
    - Very well know and commonly used (not only for front end technology). Does not execute directly in the browser, only with the use of specific development plugins like Google Web Toolkit (GWT). In GWT the Java code needs to be compiled to JavaScript. This compiled JavaScript is most often better performing compared to home-written JavaScript. But not as good as Dart.
        - Frontend usage in GWT, Errai, Vaadin, etc.
- JavaScript
    - Most of the other front end frameworks use JavaScript.
    - Large support from all kinds of browsers for this kind of JavaScript. However there are a lot of differences between browser(versions).
    - No good development support. Developing and debugging in JavaScript is complex, error prone and very time consuming.

JavaScript Frameworks

These frameworks are more complete and offer good integration capabilities.

- AngularJS
    - AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you currently have to write. AngularJS is executed within the browser, making it an good frontend for any server technology.
    - AngularJS has great integration features with CSS frameworks like Bootstrap. Combined with the Bootstrap UI module this is a great combination for seamless integration between these two frameworks. .
    - AngularJS supports:
        - Dependency Injection
        - Modules
        - Routing
        - Deep-linking
        - DOM-based templating (contrary to string-based templating)
        - Model-View-Whatever architecture
        - "Web Components" extension based upon HTML with a custom syntax.
    - Angular is build from a test automation perspective. The support for dependency injection is important. It is rather easy to build test cases for Angular.
    - AngularJS is more complex compared to Knockout or Backbone (partly due to the larger set of features and possibilities). However Angular has a large community of users and a lot of examples, questions and answers on stackoverflow.
    - Angular developers are actively involved in the development of new HTML5 standards. New versions of AngularJS will support the future standards in HTML development.
- AngularDart

    - AngularDart is the Dart implementation of AngularJS and is being created by the same team as AngularJS.
    - AngularDart is very forward. A lot of the concepts in AngularDart will be ported in the HTML5 Web Component Standards..
- Ember.js
    - Ember is functionally comparable to AngularJS.
    - Ember uses more conventions. Is more leaning towards pure HTML and is as flexible as HTML.
- KnockoutJS
    - Knockout is mainly a binding framework.
    - Unobtrusive.
- Polymer
    - Polymer is the implementation (polyfill) of new HTML5 Web Components in current standard versions of the browsers not supporting these HTML5 standards.
    - Polymer is very strict on follwing the standards. A change in these standards can result in breaking code. Polymer is still in a pre-alpha version.
- Vaadin
    - Vaadin is a development language to be used as Java code and is linked to the knowledge of the modern Java developer.
    - Combined with CSS knowledge this is a rather complete framework.
    - Vaadin uses serverside state management.
    - Vaadin uses a lot of components using boilerplating. This means the developer needs to create these components first. Java purists are very enthousiastic about this, but this might take too much time during development.
    - Vaadin works great for large data sets and supports lazy loading of data.
    - Vaadin has a large community and most components are being build in GWT. !
    - Other Play 2 and GWT

Other UI frameworks

- Oracle ADF
  Oracle ADF is a quite extensive framework consisting of both server side and client side processing. In the thin server / thin client architecture ADF uses both of these models. ADF is very suitable for building systems on top of an Oracle database or a Fusion Middleware services layer.

- Node.js
  Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.